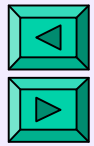
Recurrent Algorithms

Divide-and-conquer principle:

- Divide a large problem into smaller subproblems and recursively solve each subproblem, then
- Combine solutions of the subproblems to solve the original problem

Running time: by a ***recurrence relation*** combining the size and number of the subproblems and the cost of dividing the problem into the subproblems



Simple Example: $F(n) = 2^n$

- **Implicit formula:**

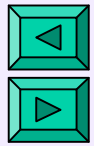
  $$F(n) = F(n-1) + F(n-1); F(0) = 1,$$

  or $F(n) = 2F(n-1); F(0) = 1; \quad n = 1,2,\ldots$

- **1, 2, 4, 8, 16, 32, 64, 128, 256, 512, … →**

  2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, …

- **Explicit formula** with $F(0) = 1$: $F(n) = 2^n$

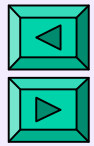


Guess and Prove an Explicit $F(n)$

Guess $F(n) = 2^n$ and prove with math induction:

- **Base inductive condition: $F(0) = 2^0 = 1$**
- **Induction hypothesis:**
  for any $n \geq 1$, if $F(n-1) = 2^{n-1}$ holds for $n-1$,
  then $F(n) = 2^n$ holds for n
  - **Proof:**

$$F(n) = 2F(n-1) = 2 \cdot 2^{n-1} = 2^n$$

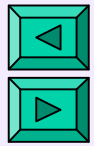


Mathematical Induction

Instead of directly proving a relationship $T(n)$, math induction proves the following conditions:

1. **Base condition** : $T(n_{\text{base}})$, e.g., $T(0)$ or $T(1)$
2. **Induction hypothesis**: for every $n > n_{\text{base}}$ if $T(n-1)$ holds for $n-1$, then $T(n)$ holds for n; or

2'. **Strong induction**: if $T(k)$ holds for every $k=n_{\text{base}}$, …, $n-1$, then $T(n)$ holds for n



Example 1.29: Fibonacci Numbers

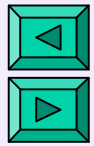
- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

  1 + 1 = 2

  1 + 2 = 3

  2 + 3 = 5

  3 + 5 = 8 …
- **Implicit formula**: $F(n) = F(n-1) + F(n-2)$
- **Analysis**: derive an **explicit formula** for $F(n)$



Characteristic Equation

- $F(n) > F(n-1)$ for all $n \geq 1 \Rightarrow$

  $2^{n-1} < F(n) < 2^n \quad \Rightarrow \quad F(n) = c\phi^n; \quad 1 < \phi < 2 \Rightarrow$

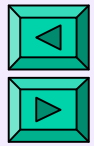
  $c\phi^n = c\phi^{n-1} + c\phi^{n-2} \quad \Rightarrow \quad \phi^2 = \phi + 1 \quad \Rightarrow$

  $\phi_{1,2} = (1 \pm \sqrt{5})/2$

- **General solution: linear combination**

$$F(n) = c_1\phi_1^n + c_2\phi_2^n$$

- $F(1)=F(2)=1 \Rightarrow F(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$



"Telescoping" of a Recurrence

- Implicit recurrence relation and its base condition (i.e., the difference equation and the initial condition):

$$T(n) = 2T(n-1) + 1;\ T(0) = 0$$

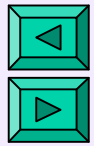
- **Closed** (explicit) form for $T(n)$ by recursive substitution of the same implicit formula:

$$T(n) = 2T(n-1) + 1$$

$$T(n-1) = 2T(n-2) + 1$$

$$\dots$$

$$T(1) = 2T(0) + 1 = 1$$



"Telescoping" ≡ Substitution

$$T(n) = 2T(n-1) + 1 \quad \text{step } 0 : \text{initial recurrence}$$

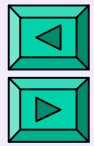
$$2T(n-1) = 2^2 T(n-2) + 2 \quad \text{step } 1 : \text{substitute } T(n-1)$$

$$2^2 T(n-2) = 2^3 T(n-3) + 2^2 \quad \text{step } 2 : \text{substitute } T(n-2)$$

$$\ldots$$

$$2^{n-1} T(1) = 2^n T(0) + 2^{n-1} \quad \text{step } n-1 : \text{substitute } T(1)$$

$$T(n) = 1 + 2 + 2^2 + \ldots + 2^{n-1} = 2^n - 1$$



Example 1.30, p.27

$$T(n) = T(n-1) + n \quad \Leftrightarrow \quad T(n) = \frac{n(n+1)}{2}$$

Closed (explicit) formula by "telescoping":

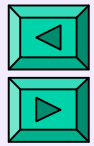
$$T(n) = T(n-1) + n$$

$$T(n-1) = T(n-2) + n - 1$$

$$\dots$$

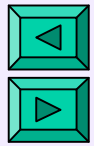
$$T(2) = T(1) + 2$$

$$T(1) = 1$$



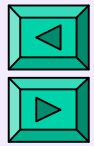
Ex.1.30: $T(n)$ by Telescoping

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= \overline{T(n-2) + (n-1)} + n \\
&= \overline{T(n-3) + (n-2)} + (n-1) + n \\
\ldots &= \overline{T(2) + 3} + \ldots + (n-2) + (n-1) + n \\
&= \overline{T(1) + 2} + \ldots + (n-2) + (n-1) + n \\
&= \bar{1} + 2 + \ldots + (n-2) + (n-1) + n = \frac{n(n+1)}{2}
\end{aligned}
$$



Ex.1.30: $T(n)$ by Guessing and Proving by Math Induction

- $T(1) = 1;\ T(2) = 1 + 2 = 3;\ T(3) = 3 + 3 = 6;$
  $T(4) = 6 + 4 = 10 \Rightarrow$ **Guessing**: $T(n) = \dfrac{n(n+1)}{2}$
- **Base condition** holds: $T(1) = 1 \cdot 2 / 2 = 1$
- **Induction hypothesis:** if the guessed formula $T(n)$ holds for $n - 1$, then it holds also for n
- ***Proof:***
$$T(n) = T(n-1) + n = \frac{(n-1)n}{2} + n = \frac{n(n+1)}{2}$$
- Thus, the guessed formula for $T(n)$ holds for all $n > 1$



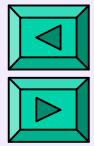
Example 1.31, p.27

- **Repeated halving principle**: halve the input in one step:

$$T(n) = T(n/2) + 1 \quad \Leftrightarrow \quad T(n) \cong \log_2 n$$

- "Telescoping" (for $n = 2^m$):

$$T(2^m) = T(2^{m-1}) + 1 \qquad T(2^2) = T(2^1) + 1$$
$$T(2^{m-1}) = T(2^{m-2}) + 1 \qquad T(2^1) = T(2^0) + 1$$
$$\ldots \qquad T(2^0) = 0$$



Ex.1.31: Explicit Formula for $T(n)$

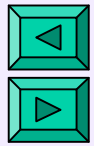
$$T(2^m) = T(2^{m-1}) + 1$$

$$= \overline{T(2^{m-2}) + 1 + 1}$$

$$\ldots = \overline{T(2^1) + 1 + 1 + \ldots + 1}$$

$$= \overline{T(2^0) + 1 + 1 + \ldots + 1 + 1}$$

$$T(2^m) = m \;\Rightarrow\; T(n) = \log_2 n$$



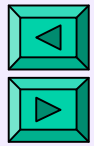
Example 1.32, p.27

- Scan and halve the input:

$$T(n) = T(n/2) + n \quad \Leftrightarrow \quad T(n) \cong 2n$$

- "Telescoping" (for $n = 2^m$):

$$T(2^m) = T(2^{m-1}) + 2^m$$
$$T(2^{m-1}) = T(2^{m-2}) + 2^{m-1}$$
$$\ldots$$
$$T(2^2) = T(2^1) + 2^2$$
$$T(2^1) = T(2^0) + 2^1$$
$$T(2^0) = 1$$



Ex.1.32: Explicit Formula for T(*n*)

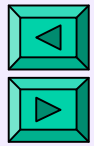
$$T(2^m) = T(2^{m-1}) + 2^m$$

$$= \overline{T(2^{m-2}) + 2^{m-1}} + 2^m$$

$$\ldots = \overline{T(2^1) + 2^2} + \ldots + 2^{m-1} + 2^m$$

$$= \overline{T(2^0) + 2^1} + 2^2 + \ldots + 2^{m-1} + 2^m$$

$$T(2^m) = 2^{m+1} - 1 \;\Rightarrow\; T(n) \cong 2n$$



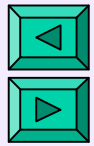
Example 1.33, p.28

- **"Divide-and-conquer" prototype;** $n \geq 2$**:**

$$T(n) = 2T(n/2) + n \quad \Leftrightarrow \quad T(n) \cong n \log_2 n$$

- "Telescoping": $\dfrac{T(n)}{n} = \dfrac{T(n/2)}{n/2} + 1; \quad T(1) = 0$

- For $n = 2^m \rightarrow \dfrac{T(2^m)}{2^m} = \dfrac{T(2^{m-1})}{2^{m-1}} + 1$



Ex.1.33: Explicit Formula for $T(n)$

$$T(2^m)/2^m = T(2^{m-1})/2^{m-1} + 1$$

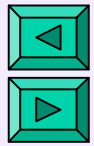
$$= \overline{T(2^{m-2})/2^{m-2} + 1 + 1}$$

$$\ldots = \overline{T(2^1)/2^1 + 1 + \ldots + 1 + 1}$$

$$= \overline{T(2^0)/2^0 + 1 + 1 + \ldots + 1 + 1}$$

$$= 0 + 1 + \ldots + 1 = m$$

$$T(2^m) = m \cdot 2^m \quad \Rightarrow \quad T(n) = n \log_2 n$$



Capabilities and Limitations

- Rough complexity analysis cannot result immediately in an efficient program but it helps to predict empirical running time of the program
- Limitations of "Big-Oh / Theta / Omega" analysis:
  - It hides the constants (e.g. c and n_0) crucial for a practical task
  - It is unsuitable for small input
  - It is unsuitable if costs of access to input data items vary
  - It is unsuitable if there is lack of sufficient memory
- But complexity analysis provides ideas how to develop new efficient methods